System Verification and Validation Plan for RoCam

Team #3, SpaceY
Zifan Si
Jianqing Liu
Mike Chen
Xiaotian Lou

November 17, 2025

Revision History

Date	Version	Notes
2025-10-21	1.0	Initial Draft
2025-10-23	2.0	Revised on section 3
2025-10-26	3.0	Revised on section 3 based on TA and peer feedback
2025-10-27	4.0	Added system tests

Contents

1 Symbols, Abbreviations, and Acronyms

symbol	description
Yolo	You only look once – an existing object detection model
CNN	Convolutional Neural Network
Jetson	abbreviatio for specifcally Jetson Nano Orin, which is the proposed board for this project
STM32	abbreviation for specifically STM32 microcontroller, which is the proposed microcontroller

This Verification and Validation (V&V) plan outlines how the RoCam project will demonstrate that the system is built correctly (verification) and fits stakeholder needs (validation). It anchors testing to the SRS, architecture (MG), and detailed design (MIS), ensuring every requirement is traceable to one or more tests. Verification relies on structured reviews, checklists, and staged testing (unit, integration, and system) to confirm functional behavior, performance expectations, interface consistency, and safety assumptions. Validation complements this with stakeholder walkthroughs, task-based inspections, and controlled field or demo sessions to confirm the system's real-world suitability and operator workflow. Roles and responsibilities are assigned to team members and the supervisor to keep eviews focused and accountable, with feedback captured and tracked. Priorities emphasize correctness, performance, reliability, and safety; lower-priority or out-of-scope items (e.g., exhaustive usability studies or independent verification of third-party components) are noted to keep the plan feasible. The document concludes with traceability tables, placeholders for unit-level details after design finalization, and appendices for parameters and optional surveys—forming a clear, practical roadmap for building credible evidence of quality.

2 General Information

2.1 Summary

The RoCam system is a ground-based camera and gimbal assembly designed to autonomously track model rockets during launch and flight. It utilizes a high-resolution camera coupled with a motorized gimbal to maintain visual lock on the rocket, providing real-time data and video feed. The system consists of a STM32 based motion controller that interfaces with the physical gimbal; a Jetson compute module that processes the camera feed; and a react frontend that displays the video preview and allows the user to control the gimbal. The primary purpose of RoCam is to replace the traditional unreliable manual tracking methods with an automated solution to produce high-quality flight data and video footage. To better assist model rocket teams across Canada to conduct successful launches by providing an affordable and reliable system that can be easily track launching movements.

2.2 Objectives

The primary objective of this Verification and Validation V&V Plan is to establish confidence in the correctness, performance, and reliability of the RoCam system by ensuring that all verification and validation activities are thorough, traceable, and aligned with project requirements. The plan emphasizes testing and peer review to confirm that functional and performance criteria are fully addressed, with improvements tracked through GitHub Issues. Validation focuses on both automated and field testing to confirm that the system performs as intended in real-world launch conditions. These qualities—correctness, performance, reliability, and traceability—are fundamental to the success of the V&V process, ensuring that the RoCam system is both technically sound and operationally dependable. Certain objectives, such as independent verification of third-party hardware and extended usability testing, are excluded due to scope and resource limitations, with reliance placed on

manufacturer and stakeholder validation.

2.3 In-Scope Objectives

Verification and validation efforts will prioritize the following qualities:

- **Correctness:** Ensure the reliability of tracking and control algorithms through testing and validation.
- **Performance:** Achieve real-time processing of 1080p 60 fps video and maintain minimal gimbal control latency.
- **Safety:** Guarantee that the system operates safely under assumed conditions, with fail-safe mechanisms.

2.4 Out-of-Scope Objectives

The following objectives are excluded from this project due to time, budget, and resource constraints:

- **Comprehensive Usability Testing:** User experience will be evaluated informally via team and client feedback rather than large-scale formal studies.
- **Third-Party Component Verification:** External hardware such as the gimbal controller and camera are assumed to perform according to their manufacturer specifications.
- **Scalability Beyond Small-Scale Rockets:** Tracking a bullet or extremely difficult targets beyond a model rocket is outside the scope of this project.

By defining these priorities and exclusions, the RoCam project ensures that available effort is directed toward validating the most critical system qualities: **performance, accuracy, and operational safety**.

2.5 Extras

Extras

Beyond the core objectives, the RoCam project includes several extras that enhance its quality and usability:

- **STM32 control circuit design:** Development of a custom control circuit to interface the gimbal actuators with the host computer, enabling precise motion control and feedback sensing.
- **User Instructional Video:** Creation of a short instructional video demonstrating system setup, calibration, and operation to support new users and ensure safe deployment.

These extras aim to improve the overall usability, accessibility, and documentation quality of the final deliverable, reinforcing RoCam's emphasis on practical integration between computer vision software, mechanical control, and end-user experience.

2.6 Relevant Documentation

Software Requirements Specification (SRS Author (2019)): The SRS document serves as a outline for the VnV plan. The SRS document contains specific functional and non-functional requirements, performance criteria, and constraints. The SRS document also contains detailed stakeholder roles and scope of the work. Aligning the VnV plan with the SRS document ensures that all specified requirements are adequately verified and validated.

Software Architecture MG (MG ?) The MG document provides details of our system architecture. To effectively design tests to validate and verify the system, it is important to understand the architecture and functionality of each module.

Software Detailed Design Document (MIS ?): The MIS document provides detailed design of our system. It provides critical information about the implementation details, including data structures, algorithms, and interfaces. This document is essential for efficient test design to specific modules.

3 Plan

This section outlines the overall Verification and Validation (V&V) plan for the **RoCam** project. It defines the structure, responsibilities, and methods used to verify that the system meets its functional and non-functional requirements and to validate that it fulfills stakeholder needs. The section covers the verification of the SRS, design, implementation, and automated testing processes, as well as the validation strategy through stakeholder reviews and field testing. Its purpose is to provide a clear, organized framework ensuring that all aspects of RoCam are systematically tested, reviewed, and proven reliable before final deployment.

3.1 Verification and Validation Team

Table 1: Verification and Validation Team Responsibilities

Name	Role	V&V Responsibilities
Zifan Si	Front End Interface Tester	Responsible for verifying and validating the usability and functionality of the react frontend interface. Ensure usability, robustness of frontend features.
Jianqing Liu	Team Lead	Conduct integration test and system validation. Validate the functionality and performance of the system against requirements and use cases. Validate system design and functionality.
Mike Chen	CV Tester	Verify and validate computer vision pipeline performance, including object detection and tracking accuracy.
Xiaotian Lou	Integration Tester and test automation developer	Implement automated testing frameworks in all software components. Implement API, and integration test across different modules.
Dr. Shahin Sirouspour	Faculty Supervisor	Provides oversight and ensures that verification and validation procedures align with academic and engineering standards. Reviews test methodologies, validates safety-critical procedures, and confirms that deliverables meet course and institutional requirements.

3.2 SRS Verification

The SRS verification will focus on verify whether the system design satisfy all functional and non-functional requirements, stakeholder needs, scope of work and constraints specified. The SRS verification will be conducted through the following steps:

3.2.1 functional and Non-Functional Requirement Review

Since the Design function will evaluate in detail whether the system align with the functional and non-functional requirements. The SRS verification will focus on verify the system design as a whole against the requirements. It will be implemented as integration test in the final stages of development.

3.2.2 Stakeholder Review

Several review sessions will be schedules with the stakeholders, such as Dr. Shahin Sirouspour, the McMaster Rocketry Team. Our team can revise the future feature implementation

based on their feedback.

3.2.3 Constraints and Scope of Work Review

Several internal review sessions will be conducted to ensure that our system design and implementation are within the defined scope of work and constraints specified in the SRS document.

Table 2: SRS Verification Checklist

Phase	**Verification Activities**
Functional & Non-Functional Requirement Review	☐ Review all requirements against current system design ☐ Create automated system tests for functional requirements ☐ Measure performance, safety, and reliability against non-functional requirements ☐ Record test results and address any failed cases ☐ Update traceability matrix after verification
Stakeholder Review	☐ Schedule review sessions with stakeholders ☐ Present verified results and collect feedback ☐ Document required changes based on feedback ☐ Obtain stakeholder confirmation or approval
Constraints & Scope of Work Review	☐ Verify implementation stays within project scope ☐ Check compliance with all technical and safety constraints ☐ Review design against timeline and budget limits ☐ Update records if scope or constraints change
Integration Verification	☐ Conduct integration tests covering all system components ☐ Validate full system behavior against SRS requirements ☐ Log and resolve all integration issues ☐ Prepare final verification report for stakeholder review

3.3 Design Verification

The design verification process for the **RoCam** project ensures that the proposed architecture, algorithms, and hardware interfaces satisfy all functional and non-functional re-

quirements before implementation. Design verification activities will be conducted through structured design reviews, peer evaluations, and the use of detailed verification checklists. The goal is to confirm that the design meets the intended performance, reliability, and safety objectives defined in the SRS.

Verification Plan

The design verification will proceed in three main stages:

1. **Internal Design Review:** Each subsystem (computer vision, control, and UI) will undergo an internal review by each team member. Each member must agree on the designed architecture, algorithms, and interfaces. Interfaces must be clearly defined for scalability.

2. **Peer Review by Classmates:** A peer design review session will be conducted with other capstone teams to obtain external feedback. Reviewers will evaluate the system's clarity, feasibility, and maintainability based on the shared design artifacts (block diagrams , interface specifications, and test plans).

3. **Supervisor and TA Review:** The project supervisor and teaching assistants will review the finalized design to ensure compliance with academic standards, capstone expectations, and safety considerations.

Verification Artifacts

Design verification will produce the following artifacts:

- Verified design diagrams and interface specifications.
- Annotated review feedback forms from classmates and supervisor.
- Updated design documents reflecting corrections and improvements.
- Completed design verification checklists (see below).

Design Verification Checklist

The following checklist will be used during design reviews to ensure completeness and consistency across all modules:

Table 3: Design Verification Checklist

Phase	**Verification Activities**
Internal Design Review	☐ Review subsystem architectures (vision, control, UI) with all team members ☐ Confirm APIs that are agreed upon ☐ Verify all module interfaces are clearly defined and documented in code ☐ Ensure design supports scalability and modularity ☐ Record review comments and update design diagrams
Peer Review by Classmates	☐ Present system design and interface specifications to peer teams ☐ Collect peer feedback on clarity, feasibility, and maintainability ☐ Address identified issues or ambiguities in documentation ☐ Record peer feedback summaries and resolutions
Supervisor and TA Review	☐ Submit finalized design package (diagrams, specs, test plans) for review ☐ Confirm compliance with academic and capstone standards ☐ Review safety considerations and risk mitigations ☐ Document supervisor and TA comments and update design documents accordingly
Verification Artifacts	☐ Store verified design diagrams and updated interface specifications ☐ Archive annotated feedback forms from peers and supervisor ☐ Maintain an updated version-controlled design document repository ☐ Complete and file the final design verification checklist

By following this structured verification plan and checklist, the team ensures that the RoCam design is both technically sound and fully aligned with its performance and safety goals prior to implementation.

3.4 Verification and Validation Plan Verification

The verification of the verification and validation (VnV) plan for the **RoCam** project will be verified through testing and peer review from peers and supervisors. Each iteration of

feedbacks and improvements will be tracked with github issues. The focus on the verification plan will ensure functionality, performance metrics are adequately covered in design.

For the validation plan, it will be measured through a combination of field tests and automated test cases. The automated test will make sure our program functions as expected in the developing environment. The field test will be used as our integration tests to validate the entire system.

Table 4: RoCam Verification and Validation (V&V) Checklist

Phase	**Actionable Verification and Validation Activities**
Verification Plan Review	☐ Conduct internal reviews of the V&V plan with team and supervisor ☐ Ensure functionality and performance metrics are fully covered ☐ Perform peer review sessions for completeness and clarity ☐ Track feedback and revisions using GitHub Issues ☐ Confirm all review comments are resolved before approval
Automated Verification Testing	☐ Develop automated tests for vision, control, and UI components ☐ Verify software behavior under simulated conditions ☐ Record test results and compare against performance criteria ☐ Update test scripts based on identified defects or regressions
Field Validation Testing	☐ Conduct full system integration tests during field trials ☐ Validate real-time tracking, communication, and safety functions ☐ Collect and analyze field performance data (latency, accuracy, stability) ☐ Log and resolve issues through GitHub and retest after fixes
Continuous Monitoring	☐ Review automated and field test outcomes after each iteration ☐ Maintain test evidence and validation summaries in repository ☐ Update the V&V plan to reflect system improvements and new test cases

3.5 Implementation Verification

The implementation verification for the **RoCam** project ensures that the developed software and integrated hardware components correctly realize the verified design. This phase confirms that the implemented system satisfies the functional and non-functional requirements defined in the SRS through a combination of structured testing, static verification, and peer review.

Dynamic Verification

Dynamic verification activities focus on confirming that the system behaves as intended under realistic operating conditions. They include the following elements:

- **Unit Testing:** Each software module (vision processing, gimbal control, data recording, and UI management) will undergo unit testing to verify functional correctness in isolation.
- **Integration Testing:** Once individual modules are verified, integration tests will confirm proper data flow between subsystems (camera input $\rightarrow$ vision algorithm $\rightarrow$ gimbal control $\rightarrow$ UI). Tests will ensure synchronization, timing correctness, and robustness under real-time constraints.
- **System Testing:** Full-system field tests will validate end-to-end performance, including real-time video tracking, system state transitions (Idle, Armed, Tracking), and video recording accuracy at 1080p 60 fps. These tests correspond directly to the verification items in the SRS functional and performance requirements.

Static Verification

Static verification will be performed throughout the implementation phase to ensure that the codebase meets quality, safety, nd maintainability standards before execution. Planned techniques include:

- **Code Walkthroughs:** Conducted during the final CAS 741 presentation, where each subsystem lead presents core implementation logic and design decisions. This serves as a semi-formal walkthrough for peer and instructor review, enabling early identification of potential design or coding issues.
- **Code Inspection:** Team members will review each other's pull requests prior to merging into the main branch. Reviews will focus on coding standard compliance, potential logic errors, exception handling, and safety-critical sections (e.g., gimbal actuation and state transitions).
- **Static Analysis Tools:** Tools such as `clang-tidy`, `cppcheck`, and `pylint` will be used to detect memory leaks, uninitialized variables, and other common implementation defects. Results from static analysis will be logged and reviewed as part of the verification record.

- **Documentation Review:** The team will verify that all inline comments, module-level documentation, and API references are consistent with the design specification and user documentation requirements.

Presentation and Usability Review

The final CAS 741 presentation will serve as both a **code walkthrough** and a partial **usability evaluation**. During the presentation, peers and instructors will observe live system operation, assess user interface clarity, and provide feedback on usability, responsiveness, and overall system behavior. This qualitative feedback will be incorporated into the final verification summary and used to refine user-facing components prior to final submission. Together, these verification activities ensure that the RoCam implementation is thoroughly tested, statically verified, and validated against both its design and real-world performance expectations.

3.6 Automated Testing and Verification Tools

The RoCam project adopts a structured automated testing approach that prioritizes maintainability, integration compatibility, and developer productivity. The selection of testing frameworks was guided by three key criteria: (1) **ecosystem compatibility**, ensuring each tool aligns with its respective language and runtime; (2) **ease of automation**, to integrate smoothly with CI/CD pipelines; and (3) **scalability**, so that test coverage can expand as the project evolves. These principles ensure a consistent and reliable testing environment across backend, frontend, and end-to-end validation stages.

Each subsystem of RoCam employs tools optimized for its function and implementation language: **Testing Frameworks:**

- **Backend (Python):** The project uses **Pytest** for unit testing due to its simplicity, readability, and powerful plugin system. It supports fixture management and integrates easily with coverage analysis and mock testing.
- **Frontend (React/TypeScript):** The project uses **Jest**, which is tightly integrated with React and TypeScript. It enables rapid component testing, snapshot validation, and ensures that UI logic behaves as expected across updates.
- **System-Level and Web UI:** The project uses **Playwright** for end-to-end (E2E) and Web UI testing. Playwright enables simulation of realistic operator workflows such as arming, tracking, and recording, ensuring smooth coordination between the frontend, backend, and physical hardware layers.

Code Coverage and Reporting Plan:
Coverage analysis is conducted using the following tools:

- **pytest-cov:** Provides granular coverage reporting for Python code, including line, branch, and function-level metrics, seamlessly integrated with Pytest test runs.

- **Jest (built-in coverage):** Tracks statement, branch, and function coverage for TypeScript components and produces standardized reports that align with Python output formats for unified tracking.

Linters and Formatters:
To maintain coding standards and prevent stylistic or logical errors, RoCam enforces consistent linting and formatting across both stacks:

- **Python:** Uses **Ruff** and **Black** for linting and code formatting to enforce PEP 8 compliance and maintain readability.
- **Frontend (TypeScript):** Uses **ESLint** and **Prettier** to maintain uniform style rules and detect syntax or logical inconsistencies.

3.7 Software Validation

Software validation confirms that **RoCam** satisfies stakeholder needs and accurately implements the intended use cases ("building the right system"). Validation complements verification and is referenced to the SRS goals nd stakeholder roles.

Stakeholder Reviews and Task-Based Inspection

- **Requirements Walkthroughs:** Scheduled reviews of the SRS and UI wireframes with the McMaster Rocketry Team (client) to confirm scope, priorities, and acceptance criteria.
- **Task-Based Inspection:** Stakeholders perform representative tasks (arm system, initiate tracking, stop tracking, export footage) while observers record issues, ambiguities, and unmet expectations.
- **Peer Reviews:** Classmate reviews focus on clarity, testability, and completeness (per CAS 741 guidance); all feedback and resolutions are logged.

Field and Demo-Based Validation

- **Rev 0/PoC Demo:** Used to validate that early functionality aligns with stakeholder expectations (e.g., preview latency, arm/track state semantics). Post-demo debrief captures change requests.
- **Supervisor Validation:** A focused session shortly after Rev 0 to confirm that the demonstrable behaviors match the SRS's product use cases and performance intent.
- **User (Operator) Sessions:** "Hallway" usability checks with camera operators validate readability in outdoor-like conditions and UI flow (single-page ops without scrolling on 1920×1080).

External Data and Bench Validation

- **Reference Footage:** Where live launches are unavailable, public model-rocket videos and internally recorded dry-runs are used to validate detection/tracking logic and reacquisition behavior.
- **Synthetic Scenarios:** Scripted clips with occlusion/smoke/backlight and varying apparent rocket sizes validate robustness claims prior to field tests.

Acceptance and Traceability

- **Acceptance Criteria:** Derived from the SRS (e.g., sustained 1080p 60 fps I/O, end-to-end latency ≤ 120 ms, successful tracking through nominal flight phases, recording with a ngle overlays).
- **Traceability:** Validation tests map to SRS goals and stakeholder needs; a table links each validation activity to the corresponding SRS item(s) and planned evidence (videos, logs, checklists).

If no suitable external datasets are available for certain scenarios, this limitation will be stated explicitly; in those cases, validation will rely on stakeholder task-based inspections and controlled synthetic footage. This section also references the SRS verification section for consistency checks between validated behaviors and specified requirements.

4 System Tests

4.1 Tests for Functional Requirements

The following manual tests are designed to evaluate the robustness of the Tracking Camera System. Each test includes its initial state, input, expected output, justification (test case derivation), and procedure. Tests are organized by functional area. The intent is to verify compliance with the functional requirements (FR) while deliberately stressing the system to expose edge cases.

4.1.1 Area of Testing: Video Acquisition and Output

test-aquire-frame: Camera Frames are Acquired at Startup

- **Control:** Manual
- **Initial State:** System powered off; camera connected; HDMI monitor attached.
- **Input:** Power on the system and allow it to fully initialize.
- **Expected Output:** Within 10s of boot, logs show that the camera is acquiring frames.
- **Test Case Derivation:** The system's operation depends on a valid video feed.
- **How Test Will Be Performed:** Boot the device; observe the program logs.

test-disconnect-camera: Camera Disconnect During Operation

- **Control:** Manual
- **Initial State:** System initialized; preview and HDMI output active.
- **Input:** Physically disconnect the camera from the system.
- **Expected Output:** On disconnect, the UI and HDMI output show a clear error message; no crash.
- **Test Case Derivation:** When the camera is disconnected, the system should gracefully handle the loss without crashing. And it should notify the operator that the camera is disconnected.
- **How Test Will Be Performed:** Disconnect the camera from the system; observe the UI and HDMI output for error messages.

test-hdmi-output: HDMI Output with Info Overlay

- **Control:** Manual
- **Initial State:** System initialized; idle state; preview and HDMI output active.
- **Input:** Move the gimbal manually through the UI.
- **Expected Output:** HDMI shows 1080p60 camera feed with a text overlay of current pan/tilt angles.
- **Test Case Derivation:** The system should be able to display the current camera feed and the pan/tilt angles for the viewers
- **How Test Will Be Performed:** Move the gimbal manually, and visually verify overlay values change smoothly and match commanded motion

test-disconnect-hdmi: HDMI Disconnect

- **Control:** Manual
- **Initial State:** System initialized; preview and HDMI output active.
- **Input:** Physically disconnect the HDMI cable.
- **Expected Output:** On disconnect, the UI show a clear error message; no crash.
- **Test Case Derivation:** When the HDMI cable is disconnected, the system should gracefully handle the loss without crashing. And it should notify the operator that the HDMI cable is disconnected.
- **How Test Will Be Performed:** Disconnect the HDMI cable from the system; observe the UI for error messages.

test-preview-real-time: Preview with Real-Time Info

- **Control:** Manual
- **Initial State:** System initialized; idle state; preview and HDMI output active.
- **Input:** Move the gimbal manually through the UI.
- **Expected Output:** Preview in the UI updates fluidly (target $\geq$15fps) and displays current pan/tilt angles.
- **Test Case Derivation:** The system should allow the camera operator to see the current pan/tilt angles and HDMI output in real-time.
- **How Test Will Be Performed:** Move the gimbal manually, and visually verify the preview in the UI updates fluidly and displays the current pan/tilt angles.

test-network-disconnect: Network Disconnect While Viewing UI

- **Control:** Manual
- **Initial State:** System initialized; preview and HDMI output active.
- **Input:** Disconnect the system from the network for 30s; reconnect.
- **Expected Output:** The system keeps functioning as normal without network; UI show a clear error message; upon reconnection, the UI session restores automatically within 5s without reboot; no data loss or crash.
- **Test Case Derivation:** The system need to be resilient against transient network failures in field conditions.
- **How Test Will Be Performed:** Manually disconnect and connect the system from the network; observe the UI for error messages.

4.1.2 Area of Testing: Recording

test-recording: Start and Stop Recording

- **Control:** Manual
- **Initial State:** System initialized; idle state; sufficient disk space.
- **Input:** Press "Start Recording", wait 10s, then press "Stop Recording".
- **Expected Output:** A 10 second 1080p60 video file is created and saved to the disk.
- **Test Case Derivation:** The system should be able to start and stop recording.
- **How Test Will Be Performed:** Press "Start Recording", wait 10s, then press "Stop Recording".

test-disk-full: Disk Full Behavior During Recording

- **Control:** Manual
- **Initial State:** System initialized; idle state; storage nearly full (simulate by filling disk leaving $<$ 1min recording capacity).
- **Input:** Start recording and allow disk to fill.
- **Expected Output:** UI show a clear "Storage Full" alert; current recording is stopped and the file is saved to the disk; no crash.
- **Test Case Derivation:** The system should be able to handle storage exhaustion gracefully.
- **How Test Will Be Performed:** Fill the disk with data until it is nearly full; start recording; observe the UI for the "Storage Full" alert; check the disk for the saved file.

test-manage-recordings: List and Download Recordings

- **Control:** Manual
- **Initial State:** At least two prior recordings exist.
- **Input:** Open recordings list; download the latest file and an older file; delete a small test clip.
- **Expected Output:** List shows correct entries (name, timestamp, duration); downloads succeed; delete removes the selected item only.
- **Test Case Derivation:** The system should be able to list, delete, and download recordings.
- **How Test Will Be Performed:** Open the recordings list; download the latest file and an older file; delete a small test clip; check the list for the correct entries; check the disk for the saved files.

4.1.3 Area of Testing: Tracking

test-manual-control: Manual Gimbal Control in Idle

- **Control:** Manual
- **Initial State:** System initialized; idle state; preview and HDMI output active.
- **Input:** Issue manual pan and tilt commands via UI.
- **Expected Output:** The gimbal follows commands smoothly.
- **Test Case Derivation:** The operator should be able to manually control the gimbal in idle state.

- **How Test Will Be Performed:** Issue manual pan and tilt commands via UI; observe the gimbal movement.

test-transition-to-tracking: Transition to Tracking When Rocket Detected

- **Control:** Manual
- **Initial State:** System initialized; idle state; preview and HDMI output active.
- **Input:** Arm the system via UI, and waving a model rocket in front of the camera.
- **Expected Output:** The system transitions to tracking state as soon as the rocket is detected; the UI shows the system is in tracking state
- **Test Case Derivation:** The system should be able to transition to tracking state when a rocket is detected.
- **How Test Will Be Performed:** Arm the system via UI, and waving a model rocket in front of the camera; observe the UI for the system to transition to tracking state.

test-tracking: Keep Rocket in Frame While Tracking

- **Control:** Manual
- **Initial State:** System initialized; tracking state; preview and HDMI output active.
- **Input:** Continue waving the model rocket in front of the camera.
- **Expected Output:** Gimbal should continuously adjust its position to keep the rocket in the frame for the duration of the test.
- **Test Case Derivation:** The system should be able to keep the rocket in the frame while tracking.
- **How Test Will Be Performed:** Wave the model rocket in front of the camera; observe the gimbal movement; observe the preview in the UI.

test-return-to-idle: Return to Idle When Rocket is Lost

- **Control:** Manual
- **Initial State:** System initialized; tracking state; preview and HDMI output active.
- **Input:** Occlude the model rocket.
- **Expected Output:** Within 2s of sustained loss, system transitions to idle state; ceases autonomous gimbal commands; UI indicates "target lost".
- **Test Case Derivation:** The system should be able to return to idle state when the rocket is lost.

- **How Test Will Be Performed:** Occlude the model rocket; observe the UI for the system to transition to idle state.

test-exit-gimbal-range: Rocket Exits Gimbal Range

- **Control:** Manual
- **Initial State:** System initialized; tracking state; preview and HDMI output active.
- **Input:** Move the model rocket outside of the gimbal's range of motion.
- **Expected Output:** The gimbal moves until it reaches its mechanical limit, then stops moving further; no errors or warnings are presented to the user; system remains responsive.
- **Test Case Derivation:** The system must safely handle objects that leave the gimbal's range without faulting or reporting spurious errors.
- **How Test Will Be Performed:** Move the model rocket outside of the gimbal's range of motion; observe the gimbal reaches its mechanical end and stops; verify that no error messages appear in the UI and that all other system functions remain operational.

4.1.4 Field Testing

Field testing will be conducted by one of our clients. The client will be provided with the manual of the system and be in charge of operating the system. The field test will be conducted in a model rocket launch event outdoors. The client will go through all the use cases of the system listed in the SRS during the field test. The use cases includes manually controlling the gimbal, arming and disarming the system, starting and stopping recording, and tracking a model rocket.

Since the field test is conducted at an external event, the exact steps of the test will vary depending on the event. The development team will be present at the event to assist the client with the test.

After the field test, the client will fill out a survey (Appendix 6.2) to evaluate the system. Additionally, the development team will review the logs to ensure the system operates as expected.

4.2 Tests for Nonfunctional Requirements

4.2.1 UI Walkthrough

The UI walkthrough will be conducted by the capstone team members themselves before the field test.

test-consistent-units: Consistent and Changeable Units

- **Type:** Manual
- **Initial State:** System initialized; idle state; preview and HDMI output active.
- **Input/Condition:** Go through the UI twice, once with metric units, once with imperial units.
- **Output/Result:** The units are consistent and changeable.
- **How Test Will Be Performed:** The capstone team members will go through the UI twice, once with metric units, once with imperial units; observe the units in the UI.

test-bilingual-ui: Bilingual UI

- **Type:** Manual
- **Initial State:** System initialized; language = English.
- **Input/Condition:** Switch UI language to French and back; navigate all UI sections, dialogs, and toasts.
- **Output/Result:** All visible strings/localized assets appear in the selected language; no truncation/overflow; no mixed-language strings; hotkeys remain functional; date/time/number formats localize correctly.
- **How Test Will Be Performed:** Perform a complete UI walkthrough in French; capture screenshots of each page/dialog; switch back to English and repeat spot checks.

test-immediate-ui-feedback: Immediate UI Feedback

- **Type:** Manual
- **Initial State:** System idle; UI ready.
- **Input/Condition:** Trigger representative actions: button press, toggle, menu selection, starting/stopping recording, arming/disarming, opening settings.
- **Output/Result:** A visible confirmation (pressed state, spinner, toast, label change) appears within 0.2s for each action.
- **How Test Will Be Performed:** Observe the UI while performing each action

test-visual-indicators: Rely on Visual Indicators

- **Type:** Manual
- **Initial State:** System initialized; speakers muted or disconnected.
- **Input/Condition:** Trigger statuses/alerts: armed, tracking, target lost, storage low/full, network disconnect, device error.
- **Output/Result:** Every status/alert is presented visually (icons, colors, banners, toasts) without relying on sound; no action-critical info is audio-only.
- **How Test Will Be Performed:** With audio disabled, induce each state/fault; verify a visible indicator appears and remains long enough to be noticed.

test-confirmation-prompts: Confirmation Prompts for Safety-Critical Actions

- **Type:** Manual
- **Initial State:** System idle; gimbal powered.
- **Input/Condition:** Attempt potentially hazardous actions: force home while tracking, hard stop, high-speed slew, firmware update during armed state, factory reset.
- **Output/Result:** A blocking confirmation dialog appears for each action; dialog text states the specific risk and outcome; default action is "Cancel"; action proceeds only after explicit confirmation.
- **How Test Will Be Performed:** Trigger each action; capture screenshots of the prompt; verify risk wording is specific (not generic); verify no motion/change occurs until "Confirm" is pressed.

test-manual-idle-mode: Manual Idle Mode / Override

- **Type:** Manual
- **Initial State:** System armed or tracking.
- **Input/Condition:** Press "Idle" (or equivalent) control while tracking.
- **Output/Result:** System transitions to idle within 1 s; autonomous commands cease; manual controls become available; status clearly shows "Idle."
- **How Test Will Be Performed:** Enter tracking, then invoke manual idle; observe immediate cessation of gimbal motion; verify state banner change and manual jog controls responsiveness.

test-no-collection-pii: No Collection of PII

- **Type:** Manual
- **Initial State:** System initialized; default configuration.
- **Input/Condition:** Inspect all settings, logs, exported files, filenames/paths, and network-configurable fields.
- **Output/Result:** No fields request or store personal identifiers (names, emails, phone numbers, precise home addresses, faces with identity tags); logs contain only technical data; telemetry excludes PII.
- **How Test Will Be Performed:** Open each settings and log panel; download logs/recordings; inspect metadata (EXIF/JSON/CSV) for PII keys; verify any optional labels are device/session IDs, not personal data.

test-color-vision-deficiency: Color-Vision Deficiency Accessibility

- **Type:** Manual
- **Initial State:** System initialized; all status indicators visible.
- **Input/Condition:** Review all status/alert states (normal, armed, tracking, warning, error) and interactive controls (enabled/disabled) under simulated Protanopia/Deuteranopia/Tritanop or printed CVD reference swatches.
- **Output/Result:** Each state is distinguishable without relying on hue alone (uses shape, text labels, patterns, or contrast); contrast ratio $\geq$ 4.5:1 for text/icons against background.
- **How Test Will Be Performed:** Cycle UI through all states; verify distinct icon shapes/labels; optionally view via a CVD simulator; record pass/fail with screenshots; confirm tooltips/textual labels exist for each color state.

4.2.2 Code and Documentation Walkthrough

test-manual-arm-mode: Manual Arm Mode Only

- **Type:** Static
- **Initial State:** N/A
- **Input/Condition:** Inspect all state transition code and configuration defaults.
- **Output/Result:** The system only enters armed mode when explicitly commanded by the user, never automatically.
- **How Test Will Be Performed:** Review code for all references to the armed state, verify guards require user input.

test-gimbal-interface: Gimbal Interface Documentation Review

- **Type:** Static
- **Initial State:** N/A
- **Input/Condition:** Review the interface specification with the client, including command set, telemetry, timing, and safety notes.
- **Output/Result:** The client confirms the interface documentation is complete, clear, and suitable for supporting multiple gimbal systems.
- **How Test Will Be Performed:** Conduct a walkthrough meeting with the client, check all required topics against a prepared checklist.

4.2.3 Error Handling and Logging

test-report-errors: Report All Errors to the User

- **Type:** Manual
- **Initial State:** System initialized and idle.
- **Input/Condition:** Manually trigger hardware, communication, and software faults.
- **Output/Result:** Each error generates a visible and descriptive message on the user interface.
- **How Test Will Be Performed:** Disconnect devices, kill processes, and simulate faults while observing the UI for clear error reporting.

test-stop-immediately: Stop Immediately on Unrecoverable Error

- **Type:** Manual
- **Initial State:** System operating in tracking mode.
- **Input/Condition:** Introduce a fatal condition such as gimbal communication loss or corrupted control process.
- **Output/Result:** The system immediately stops all automated operations and transitions to a safe idle state.
- **How Test Will Be Performed:** Disconnect the gimbal or crash the tracking process, verify that motion stops and the system displays a clear unrecoverable error message.

test-validate-commands: Validate All Commands

- **Type:** Manual
- **Initial State:** System idle.
- **Input/Condition:** Enter invalid or out-of-range user commands through the UI or API.
- **Output/Result:** The system rejects invalid commands and displays a validation error without sending them to the gimbal.
- **How Test Will Be Performed:** Attempt to send commands beyond gimbal limits or in unsupported formats and confirm that they are blocked with clear feedback.

test-log-operations: Log All Operations

- **Type:** Manual
- **Initial State:** System initialized and user logged in.
- **Input/Condition:** Perform a series of user operations such as arming, tracking, and recording.
- **Output/Result:** Each action is logged with a timestamp and operation details in the system log.
- **How Test Will Be Performed:** After the field test, review the log file to confirm all actions and timestamps are recorded accurately.

4.2.4 Field Testing

test-user-survey: User Survey

- **Type:** Static
- **Initial State:** N/A
- **Input/Condition:** N/A
- **Output/Result:** Completed user survey.
- **How Test Will Be Performed:** After the field test, the client fills out the survey and returns it to the development team.

4.3 Traceability Between Test Cases and Requirements

Test ID	Requirements
test-aquire-frame	FR-1, SLR-2
test-disconnect-camera	FR-1
test-hdmi-output	FR-8, FR-9, INT-1, SLR-3
test-disconnect-hdmi	FR-8
test-preview-real-time	FR-10
test-network-disconnect	FR-10
test-recording	FR-11
test-disk-full	FR-11
test-manage-recordings	FR-12
test-manual-control	FR-2, FR-6
test-transition-to-tracking	FR-3, FR-4
test-tracking	FR-2, FR-3, FR-7
test-return-to-idle	FR-5
test-exit-gimbal-range	FR-2, FR-7
test-consistent-units	EZ-1, PI-1
test-bilingual-ui	PI-2
test-immediate-ui-feedback	EZ-2
test-visual-indicators	EZ-3
test-confirmation-prompts	EZ-4, UPR-3
test-manual-idle-mode	SCR-1
test-no-collection-pii	LR-1
test-color-vision-deficiency	AR-1
test-manual-arm-mode	SCR-4
test-gimbal-interface	INT-2
test-report-errors	RFR-1
test-stop-immediately	RFR-2
test-validate-commands	IR-1
test-log-operations	AUR-1
test-user-survey	AR-1, AR-2, SR-1, LR-1, UPR-1, UPR-2, PAR-1, SLR-1, RFR-5, CR-1, CR-2, SCR-2, SCR-3, RFR-3, EPE-1

5 Unit Test Description

5.1 Unit Testing Scope

5.2 Tests for Functional Requirements

5.2.1 Module 1

1. test-id1

   Type:
   Initial State:
   Input:
   Output:
   Test Case Derivation:
   How test will be performed:

2. test-id2

   Type:
   Initial State:
   Input:
   Output:
   Test Case Derivation:
   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

5.3.1 Module ?

1. test-id1

   Type:
   Initial State:
   Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

References

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 User Survey Questions

Please answer each question based on your experience during the field test. For questions rated on a scale, circle the number that best reflects your opinion: 1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, 5 = Strongly Agree.

User Interface

1. The interface was readable and easy to see outdoors. (1 2 3 4 5)
2. The interface looked professional and well-designed. (1 2 3 4 5)
3. The important information was immediately visible on the screen. (1 2 3 4 5)
4. No unnecessary or confusing information was shown. (1 2 3 4 5)
5. The system was easy to learn and operate after a short time. (1 2 3 4 5)
6. The terms and labels used in the interface were clear and understandable. (1 2 3 4 5)

System Performance

1. The tracking footage was stable and smooth. (1 2 3 4 5)
2. The tracking system maintained lock on the rocket reliably. (1 2 3 4 5)
3. The system operated as expected without major errors. (1 2 3 4 5)
4. Once armed, the system operated fully autonomously. (1 2 3 4 5)
5. I was able to fully operate and monitor the system remotely. (1 2 3 4 5)

User Experience and Feedback

1. Were any colors, symbols, or content elements disrespectful or inappropriate to you? *(Please describe if applicable)*

2. What improvements would you suggest for future versions of RoCam?

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

Jianqing Liu

1. **What went well:** I was responsible for the Part 4 of this deliverable, focusing on all the system tests. I think it was really easy to come up with all the test cases because We already completed the SRS.

2. **Pain points and resolution:** I think some of the test cases are very verbose. For some of the tests, how test will be performed is just the same as the input and output condition of the test case.

Mike Chen

1. **What went well:** I contributed primarily to Part 3 of this deliverable, focusing on the verification and validation plan and the automation tools section. The structure

and clarity of this part improved substantially after integrating peer and supervisor feedback. I ensured that each verification activity was directly tied to the SRS and our test framework.

2. **Pain points and resolution:** I initially faced several issues with GitHub CI configuration—some commits overwrote teammates' work and caused LaTeX compilation failures. I resolved these by restoring the affected commits, redesigning the YAML workflows, and setting up branch protection rules. Revisions based on feedback also required extensive reformatting to achieve consistency across sections.

3. **Knowledge and skills to acquire:** I plan to improve my skills in CI/CD automation and verification planning, particularly integrating automated builds, test execution, and documentation generation through GitHub Actions.

4. **Approaches to acquire skills:** I will (1) study advanced GitHub Actions pipelines from open-source projects and (2) create a smaller test repository to simulate multi-stage verification workflows. I selected the second approach because it provides hands-on experience with merge control, build automation, and verification feedback integration.

Frank

1. **What went well:** I contributed to both Part 2 and Part 3 of the deliverable, ensuring that the design and verification sections were fully aligned with the SRS. My work focused on refining the test structure, verifying requirement traceability, and improving the flow between verification items and their corresponding system requirements.

2. **Pain points and resolution:** The main challenge was maintaining consistent terminology and depth of explanation between sections written by different team members. To resolve this, I reviewed the SRS and cross-checked every requirement reference to ensure correctness and coherence across parts.

3. **Knowledge and skills to acquire:** I plan to strengthen my knowledge of requirements traceability and automated testing frameworks, particularly Jest and Playwright, to improve coverage and maintain consistency with system goals.

4. **Approaches to acquire skills:** I will (1) review documentation on structured traceability mapping and (2) develop example UI and integration tests that link directly to specific SRS requirements. I chose the second approach because applying traceability in practice will reinforce both documentation and test alignment.

Xiaotian Lou

1. **What went well:** I focused mainly on Part 2 of the deliverable, organizing the SRS and design verification sections. The verification checklists and phase structure improved document readability and made the review process more systematic.

2. **Pain points and resolution:** The primary difficulty was harmonizing writing styles and section formatting across contributors. I standardized tables, checklists, and layout conventions to produce a consistent and professional presentation.

3. **Knowledge and skills to acquire:** I aim to gain more experience in formal verification documentation and structured validation reporting to improve the traceability and completeness of future deliverables.

4. **Approaches to acquire skills:** I will (1) study professional verification and validation templates from engineering projects and (2) apply checklist-based verification methods in upcoming development stages. I selected the first approach to strengthen my understanding of documentation standards and best practices.